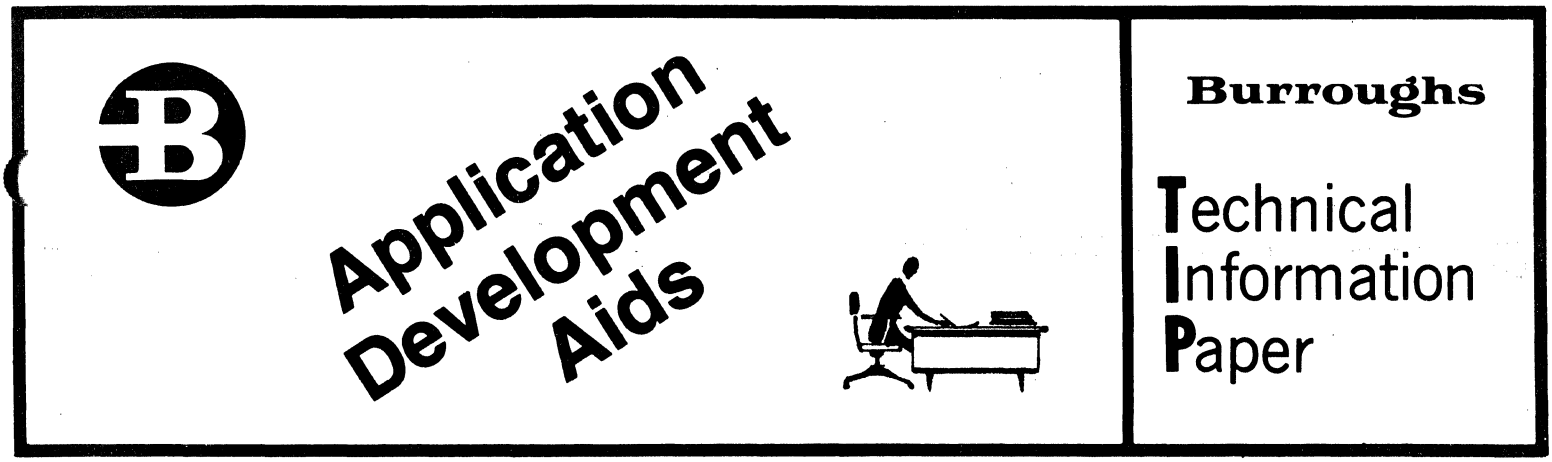

DATE: July 20, 1983

Author: Fred Trout
TSC-WEST

PRODUCT: DMSII (all systems)

SUBJECT: PHYSICAL ATTRIBUTES FOR INDEX SEQUENTIAL STRUCTURES

This paper will develop the criteria and specific recommendations for choosing appropriate values for the physical attributes of index sequential sets and subsets. It includes formulas and algorithms which differ from previously released suggestions. The characteristics and phenomenon encountered in set insertion are also discussed.

The attributes discussed are:

POPULATION or MAXENTRIES
FAMILYNAME or PACK
LOADFACTOR or SPLITFACTOR
BLOCKSIZE
AREALENGTH or AREASIZE
AREAS
BUFFERS

Copyright © 1983 Burroughs Corporation

TO: BMG Communications Codes DOC 2 and DOC 4

Printed in U.S. of America

1114238-018

```
TIP+TIP+TIP+TIP+TIP+TIP+TIP+TIP+TIP+TIP+TIP+TIP+TIP+TIP+TIP+TIP+TIP
T                                                   I                   P
T    Burroughs                                      I                   P
T        Western Region                             I   Technical       P
T            Technical Support Center               I   Information     P
T                 Product Support                   I   Paper           P
T                                                   I                   P
TIP+TIP+TIP+TIP+TIP+TIP+TIP+TIP+TIP+TIP+TIP+TIP+TIP+TIP+TIP+TIP+TIP
```

Date: June 22, 1982

Author: Fred Trout
TSC-WEST

Product: DMSII (all systems)

Subject: PHYSICAL ATTRIBUTES FOR INDEX SEQUENTIAL STRUCTURES

This paper will develop the criteria and specific recommendations for choosing appropriate values for the physical attributes of index sequential sets and subsets. It includes formulas and algorithms which differ from previously released suggestions. The characteristics and phenomenon encountered in set insertion are also discussed.

The attributes discussed are:

POPULATION or MAXENTRIES
FAMILYNAME or PACK
LOADFACTOR or SPLITFACTOR
BLOCKSIZE
AREALENGTH or AREASIZE
AREAS
BUFFERS

PHYSICAL ATTRIBUTES
INDEX SEQUENTIAL SETS and SUBSETS

This document develops the pragmatics of Burroughs DMSII DASDL capability to define the physical characteristics of files which contain the access structures for accessing data sets using the Index Sequential technique.

INTRODUCTION

The value specified for an index file's physical attributes will determine the amount and type of resource consumed. Their interdependent effect is critical to the overall performance of the system. Processor, memory, I/O time, I/O contention, and disk space are all affected by the value of a index structure's physical attributes. In some cases, the value of one attribute affects the criteria for the value of another, or the delta value of an attribute will have a positive effect on one resource and negatively affect others. An understanding of these trade-offs must be developed by the database designer.

The most important criteria for choosing the set's physical attributes is the population of the referenced records. However, the designer must also consider the effects that are dependent on its usage, i.e., the access type (random, sequential); the function mode (inquiry, update); the file volatility (adds, deletes); and the number of transaction processors that have the database open.

Each site or application will have different criteria for determining the value of these interdependent resources and the efficiency characteristics of the application systems. Each site or application will have different requirements for performance and capability.

If the site has an abundance of physical resources and the application system has simple performance requirements, then the DBMS designer can approach DMSII's physical attribute specification as a "black box". Using the DASDL defaults will give a valid index structure that will work within certain limitations, but it is not likely to be the most efficient description for that set's usage criteria. Even then, the actual capability to insert an expected population may be significantly reduced. For instance, on Medium Systems, the casual observer might expect the defaults to develop an index file size for a population of 200,000 entries. In actual loading, however, the file limits will be exceeded about 196,000 for a sequential load and 100-120,000 for a random load. Large and Small Systems have similar differences for the random load case.

This simply points out the need for careful consideration of the physical attributes even if the requirement for efficient characteristics is not paramount. Thus the following discussion will attempt to describe the practical application of the values for Index Sequential structure's physical attributes. The examples and recommendations given are for the purpose of avoiding the most common misunderstandings of the nature of this accessing technique. For any

recommendation, a specific application's requirements or site's resources might negate the appropriateness of that suggestion. It remains the responsibility of the DBMS designer to measure the trade-offs between the resource consumption and operating characteristics for a given system.

POPULATION and MAXENTRIES

REALISTIC DATA SET POPULATIONS

Since most of the resource-critical attributes are dependent on the number of expected entries, it is important to begin with an accurate estimate of the population of the data set or included records. A "realistic" population is not grossly inflated by long range estimates or worst-case possibilities. It may, however, contain a minimum amount of "safety factor" inflation for the short-term, a period of time that could be as close as the next DASDL reorganization. There are more appropriate means of developing room for unexpected growth than inflating the population.

MAXENTRIES

The DASDL syntax feature for defining file size with a MAXENTRIES or POPULATION clause is not recommended. Experience has shown that this technique is misused more than correctly used. Its proper use requires a detailed understanding of the default values of the other attributes and their interaction. It is recommended that a comment about the designed capacity and the probable population for each level be included in the set's description. The file's size should be determined by the BLOCKSIZE, AREALENGTH, and AREAS attributes. On Medium and Large Systems, the MAXENTRIES value includes entries "used" by the root and coarse tables and the header block.

FAMILYNAME

The criteria for establishing the media location for the index files are too dependent on site, application, and real-time usage parameters for a reasonable recommendation in this discussion. There are, however, a few subtle criteria the DBMS designer may wish to consider:

Spanning sets and automatic subsets can be GENERATED by the DMSII system. Protection from certain media failures can be designed by defining the index files to be on a different media than the data sets.

Tests of index files on a fast media (5N disk) have shown significant improvement in throughput for random access.

Since the I/O's to a set are completed before the I/O to the data set, no I/O overlap optimization via media selection can be designed for the relationship. However, for extremely limited usage and sequential searching, some arm-movement time might be reduced by separating the structures.

LOADFACTOR (SPLITFACTOR on Small Systems)

GENERAL LOADFACTOR CONSIDERATIONS

Perhaps the most misunderstood attribute of all, LOADFACTOR has a significant effect on resource consumption, much more effect than a basic understanding might imply.

The concept of loadfactor was first introduced to provide space in a fine table during a sequential file load application in order to reduce the probability of table overflow during subsequent inserts. This reduced excessive I/O on extended binary searches. Loadfactor was retained even after binary-tree-structured index sequential techniques were introduced to reduce the probability of table-splitting during subsequent inserts. Early algorithms for table-splitting were very restrictive and slow. Current algorithms have reduced this impediment also.

Technology in data base design, increased use of on-line access, and the requirement for multiple access structures have exploded the amount of random access usage. The LOADFACTOR attribute in a random insert environment will have a critical effect on performance and capacity. These effects are not obvious to the casual observer, so the following detailed discussion seems necessary.

DATA SET LOADING

Data set loading may be defined as an application program that creates (or greatly extends) a data file by functional CREATE/STORE, adding new records to the data set, and therefore requiring inserts into a set or sets.

The simple case of a sequential load is the creation of records in the same sequence as a single spanning set. In this case, the inserts and table-splitting happen in an orderly fashion. Each split will leave LOADFACTOR percent entries in the "previous" table and continue inserting into a "following" table. The coarse tables are similarly produced. Applications which add records where the key has the characteristic of increasing in value will also develop the index tables in a similar fashion. High value LOADFACTORS (90 to 99 %) are quite appropriate. It results in a high effective loading and a minimum number of tables.

If the data set has multiple sets, then, obviously, only one can be loaded sequentially; the other set's inserts will be random. A random insert will require finding a slot in a table at the proper sequence, and thus require a search of the structure. This, of course, is much slower than a sequential load. It uses much more processor time and requires many more I/Os.

FAST SET GENERATES

Available on Burroughs DMSII is a technique for loading data sets that is typically faster for large populations and many data sets and/or multiple sets. For the generation of spanning sets and automatic subsets, they use an algorithm that reads the data set, producing a file of keys and data set pointers, sorting it in key sequence, building the fine tables, and likewise, the coarse and root tables.

The execution of this technique involves a DASDL compilation describing only the data sets, a load of those data sets, a DASDL REORGANIZATION which includes the sets, and the generation of the sets described above. Since the generate will develop the tables according to LOADFACTOR as in a sequential load and the LOADFACTOR used for subsequent adds may differ in value (the reasons are developed later), an additional DASDL UPDATE or other change to LOADFACTOR may be required.

This technique has been extremely successful in several applications. Time reductions by a factor of 22 have been reported. Several undesirable factors must also be considered. These include the DASDL compilation time, the data base unavailability, and the fact that the generation is not audited causes a discontinuity.

RANDOM SET INSERTIONS

As noted before, random accessing, specifically, random inserts are becoming the more typical processing technique. This increased usage has focused more attention and research into the effects of this type of access, and some new insights and recommendations have been formed.

RANDOM TABLE-SPLITTING

The table-splitting algorithm, although it is the same as used for sequential inserts, has a entirely different effect on actual entries loaded per table. Therefore, the file's capacity, levels, etc. (all the variable resource consumers), are affected.

For the following examples, a table size of 100 entries will be used to avoid the confusion between LOADFACTOR, (expressed in percent) and the number of entries. The DASDL compiler will translate this percent into some integer number of entries.

The table-splitting algorithm breaks the table into three areas: from entry 1 to entry(100-LOADFACTOR) on the left end; from entry(LOADFACTOR) to 100 on the right; and the difference in the middle. LOADFACTOR of 50 will have no middle area. The number and location of the entries moved (split) to the new table is dependent upon the location of the requested insert. If the location is within the left or right area, then that area is moved. Otherwise, the entries from the requested location to the nearest end of the table are moved to the new table.

This algorithm, for high (or low) LOADFACTORs, tends to produce new tables with a small number of entries and, therefore, a small range of values for subsequent inserts. The large (old) table will tend to recieve more of the new entries and therefore, become full, split, and propagate more (nearly empty) tables. This phenomenon results in many tables with a few entries each, thus lowering the average actual entries per table, reducing the actual capacity, increasing the number of tables generated, and perhaps increasing the number of levels.

As the LOADFACTOR approaches 50%, the propagated tables include more entries thus reducing this negative effect. A simulation of this algorithm showed the following (selected) values for a population of 50,000 and tablesize of 100:

LOADFACTOR =	50%	66%	80%	95%
AVERAGE # OF ENTRIES/TABLE =	69	64	57	51
# OF FINE TABLES BUILT =	720	777	871	981

Note: see appendix for an expanded range of loadfactors.

The values for AVERAGE # OF ENTRIES/TABLE can be used for percentage values in the following formula. The most effective LOADFACTOR of 50% (69-70% actual loading) has been studied by several academicians. In 1978, A. Yoa published proof that the storage utilization is approximately the natural log of 2 (69%) for B-trees of high orders. (ACTA INFORMATICA, VOL.9). The table-splitting algorithm used by Yao is similar to the DMSII algorithm for a 50% LOADFACTOR.

The same phenomenon controls the effectiveness of the coarse tables, and that fact has a geometrical effect in determining the actual capacity of each level. The theoretical formula for determining the fine table CAPACITY required for a given population and number of levels is :

```
CAPACITY =  POPULATION / (A ** (L - 1))

  where A = average # of actual entries per table
                expressed as a decimal percentage
  and   L = # of levels (root level = 1)
  EXAMPLE:
      POPULATION = 40,000 (referenced records)
               A = 0.69 (for a 50% LOADFACTOR)
               L = 3 (root, 1 coarse, and fine levels)

         = 40000 / .69 **(3-1)
         = 40000 / .4761
         = 84,016 entries
```

That is to say, in order to (reasonably) insure staying under four levels when randomly adding 40,000 records, the set should be designed to contain 84,000 entries at the fine level.

BINARY GENERATIONS OF TABLES

Another phenomenon of random loading with a 50% LOADFACTOR exists. For any given level of tables, the "first" full table would have a average actual loading of 100%. When it splits, the loading would average 50%. The two tables would the continue to fill until the average approaches 100%, then they would split, and the average becomes 50% again. The four tables become full and split to eight half-full tables, and so on for each BINARY GENERATION. Tracing the average actual loading thru a range of populations would theoretically produce a sawtooth curve, dropping sharply from 100% to 50%, building slowly back to 100%. Small populations are particularly affected by this phenomenon.

Actual random loading being "randomly random" causes this curve to dampen rather quickly, (around 100 tables), and its range becomes 66 to 73% with jagged nodes between the vertexes. Unless empirically proven to the contrary, the prudent DBMS designer would choose to use a lower "A" value (0.66) in the capacity formula in order to anticipate the possibility of the designed population creating a "low" average.

On Medium and Small Systems, the average actual loading may be affected by long strings of duplicates. The duplicates will be added to one end of the list and therefore develop the tables where the actual loading will equal LOADFACTOR. On Large Systems this effect will only happen when DUPLICATES FIRST or LAST is specified.

Higher (or lower) LOADFACTORs are less affected by this phenomenon and stabilize within the first few splits to a constant, but lower, average actual entries. See APPENDIX.

BLOCKSIZE

There are three critical, interdependent physical attributes that provide the basis for the relationship between performance and resource consumption. These are LOADFACTOR, which is likely to be 50% for random adds or high (90-99%) for sequential adds; BUFFERS, which will be discussed later; and BLOCKSIZE or TABLESIZE, i.e., the number of entries per index sequential table. Again there are major differences between sequential and random usage.

SEQUENTIAL ACCESS

Differences in BLOCKSIZE of a set that is accessed sequentially usually has only a small effect on the overall performance. This is due mainly to the ratio of physical I/Os to the set versus the data set. The value of this ratio is the average actual loading developed by LOADFACTOR and type of creation. In a properly developed set, the set I/Os are usually not significant.

Typically the BLOCKSIZE criteria has more weight placed on memory usage. Since sequential access will use only the fine tables, even the number of levels is not of prime importance unless significant random access is required. The major criteria becomes the buffer size, the number of buffers, and the possible effects of readahead, i.e. memory usage, on one hand; and the number and effect of I/Os on the other. One might consider changing the set type to ordered-list.

RANDOM ACCESS

Using the developed formula for the capacity of fine table entries, the minimum BLOCKSIZE needed to provide the desired characteristics is easily derived. Given a realistic population, (P); an average actual loading (derived from the chosen LOADFACTOR), (A); and the maximum number of levels developed to contain this population, (L); the BLOCKSIZE in entries is the (L)th root of the CAPACITY, (C).

```
BLOCKSIZE = C ** (1 / L)
      where C = P / (A ** (L - 1))
```

The DBMS designer may wish to increase this derived BLOCKSIZE by a small number of entries to better utilize disk sector boundaries after the blocksize in bytes is calculated from entry size. However, this increase will affect buffer (memory and I/O) resources.

Since the average actual loading value for a 50% LOADFACTOR and a 66% LOADFACTOR are both recommended to be 0.66 and its accuracy is highly dependent on the actual population and randomness of the added keys, an approximate, shortcut formula can developed for specific level requirements:

For two levels:
the square root of 1.5 times the population,
BLOCKSIZE = (1.5 * POP) ** (1 / 2).

For three levels:
the cube root of 2.3 times the population,
BLOCKSIZE = (2.3 * POP) ** (1 / 3).

For four levels:
the fourth root of 3.5 times the population,
BLOCKSIZE = (3.5 * POP) ** (1 / 4).

AREALENGTH (AREASIZE)

The AREALENGTH attribute is the typical variable that determines the file's size. The AREAS attribute is the other determinate, but its value is limited and often fixed for strategic reasons. The AREALENGTH attribute determines the size of the physical containers of disk allocated as file file expands. It is recommended that the value be expressed in BLOCKS or TABLES as that is the smallest physical container.

There are several uses of adjusting AREALENGTH for optimizing a given environment. A large value which could contain the whole population in one area may reduce seek time in a single user (pack and file) environment. In a high volume, multiple-user environment, smaller areas that would increase the number of areas for a given population, could be spread over several spindles, thus, increasing I/O overlap possibilities. Disk allocation must also be considered; small areas might checkerboard available disk, large areas need to find contiguous disk space and vice versa.

In order to insure a file size large enough to contain index tables for given population of data set records, the population in blocks or tables must be estimated from known or derived attributes. The total number of tables in the file is the sum of tables at each level. The number of tables at a level is calculated by the population of the next higher level divided by the average actual loading expressed in entries. The root level by definition is a single table. Mathematically expressed, the algorithm is:

where B = BLOCKSIZE of the index tables,
and A = average actual loading (specified, derived from LOADFACTOR, or empirically determined) expressed as a decimal percentage,
then Ea (the average actual loading in entries) = B * A.

then where P = the expected population of the data set (or subset),
and L = the required number of levels designed,

$$T \text{ (total tables)} = 1 + \sum_{n=2}^{n=L} \left(P \,/\, (Ea \;**\; (n - 1)) \right).$$

Example:

```
P = 10,000
B = 43
A = 0.66
L = 3
     Ea = 43 * 0.66 = 28.38
     for n = 2
        Tn = 10000 / 28.38 ** (2 - 1) = 353
     for n = 3 = L
        Tn = 10000 / 28.38 ** (3 - 1) = 13
T = 1 + 13 + 353 = 367 tables
```

Once the approximate number of tables are known, the AREASIZE can be determined from the requirements of disk allocation criteria.

AREAS

The AREAS and AREALENGTH attributes are the factors that determine the capacity of the index file. They must specify a size that is large enough to contain not only the expected number of tables, but some safety factor amount of additional space. Unexpected high populations or unusually skewed random distributions might increase the number of tables or even the number of levels. These attributes are the proper place for this safety factor since the allocated disk resource will only be used if it is needed. The size of the safety factor should be large enough to enable the DBA to schedule a reorganization without causing a major disruption.

All the Burroughs systems provide an expandable container for the physical disk addresses of each allocated area. If the AREAS attribute is small then the memory container is smaller. For databases containing a large number of physical files, this savings can be significant. On Small and Medium Systems, the limit for AREAS that will increase the minimum size is 25 areas; Large Systems, 20 areas.

It follows then, that if the DBMS designer has no overriding disk allocation criteria, the choice of defining 25 AREAS and putting the expected tables into 80% of that limit (20 areas) would be an appropriate method. In the example before, dividing 367 tables (+1 for header block on Medium and Large), by 20 areas gives an AREALENGTH value of 19 BLOCKS.

BUFFERS (Medium and Large Systems only)

The BUFFERS attribute determines a trade-off between I/O and memory. Too few buffers for a given enviornment results in unnecessary I/O, too many results in excessive memory requirements. It describes a buffer pool for each file that may be sensitive to the amount and type of usage. The "system" parameter allocates a number of buffers for the file regardless of usage. The USER or RANDOM USER parameter adds buffers to the pool for each non-sequential user, and SERIAL is for, of course, sequential users. SERIAL values of two or more initiate READAHEAD algorithms.

For the typical environment, a random access will require only one buffer at a time as the search follows the levels of tables. Since all random users will access the root table, this buffer can be provided by a "system" value of 1. The sequential user will be searching only through the fine tables, again, using one buffer. A BUFFERS specification of 1 + 1, or 1 + 1 + 2 if READAHEAD is desired, seems appropriate for most batch environments.

There exists, however, many usage environments where this provides too many or too few buffers. To try to delimit the criteria for all those environments would be beyond the scope of this discussion. One common environment where 1 + 1 PER USER is not appropriate can be discussed. For a low volume request-rate and a large number of users, typical of a volatile online system, many unproductive buffers may be allocated. A smaller total pool may be defined by using a X + 0 specification allowing N (on-line) users to share X buffers.

The BUFFER attribute in no way limits or demands that a buffer pool be of the exact size defined. The access routine will go above the buffer limit when necessary for certain internal requirements, or deallocate infrequently used buffers below the calculated limit. On large systems, the access routines have refined this management to a point where a specification of 0 + 0, allowing the ACR total management, has been very successful for volatile environments.

SUMMARY

This discussion has hopefully provided some insight into the need and criteria for carefully choosing the physical attributes for index sequential sets. Included were some different perspectives for the priority of criteria and some different formulas for specific values. They may, in some cases, expand or disagree with similar information found in the several DMSII manuals or classes. In given environments, either (or neither) recommendation could work.

APPENDIX

SIMULATED EFFECT OF LOADFACTOR ON RANDOM LOADS

POPULATION = 50,000 / TABLESIZE= 100

ENTRY INSERT LOACTION DEPENDENT RESULTS:

LOAD FACTOR	AVERAGE # ENTRIES	# OF TABLES	SMALLEST TABLE
5	51.760	966	6
10	52.798	947	11
15	54.230	922	16
20	58.072	861	21
25	60.901	821	26
30	62.814	796	31
35	65.104	768	36
40	65.789	760	41
45	67.843	737	46
50	69.444	720	51
55	68.027	735	46
60	66.934	747	41
65	64.683	773	36
70	62.893	795	31
75	59.809	836	26
80	57.405	871	21
85	54.645	915	16
90	53.648	932	11
95	50.968	981	6

CONCLUSIONS:

A 50% LOADFACTOR will produces the highest average table loading and lowest table population for random loads (file additions via Create-Stores). As the LOADFACTOR increases (or decreases) the loading becomes less effective.

NOTE: Generating the set via REORGANIZATION will use the exact LOADFACTOR for table loading. To produce a equal number of tables, use 70%; for maximum compression, use 99%; then UPDATE back to 50% for normal processing.

Table of Contents

PHYSICAL ATTRIBUTES 1
INTRODUCTION 1
POPULATION and MAXENTRIES 3
REALISTIC DATA SET POPULATIONS 3
MAXENTRIES 3
FAMILYNAME. 4
LOADFACTOR (SPLITFACTOR on Small Systems) 5
GENERAL LOADFACTOR CONSIDERATIONS. 5
DATA SET LOADING 5
FAST SET GENERATES 6
RANDOM SET INSERTIONS. 6
RANDOM TABLE-SPLITTING 6
BINARY GENERATIONS OF TABLES. 8
BLOCKSIZE 9
SEQUENTIAL ACCESS. 9
RANDOM ACCESS. 9
AREALENGTH (AREASIZE) 11
AREAS 12
BUFFERS (Medium and Large Systems only) 13
SUMMARY 14
APPENDIX. 15
SIMULATED EFFECT OF LOADFACTOR ON RANDOM LOADS 15